Lecture 3: Hardness Amplification

Topics in Pseudorandomness and Complexity Theory (Spring 2017)
Rutgers University
Swastik Kopparty
Scribe: Amey Bhangale

In this lecture, we will see starting with a *hard* function for circuits of large size, how to construct even *harder* function.

1 Hardness Amplification

The general theme in hardness amplification is given a function which is hard for a computational model, can we construct even *harder* function.

Recall the definition of worst-case hard and δ-hard functions for circuits.

Definition 1 (worst-case hard)**.** *A function f is said to be worst case hard for circuits of size s if for all circuits of size at most s there exits $x \in \{0,1\}^n$ such that $C(x) \neq f(x)$.*

Definition 2 (average-case hard)**.** *A function $f : \{0,1\}^n \to \{0,1\}$ is called δ-hard for circuits of size s if for any circuit C of size at most s*

$$\Pr_x[C(x) = f(x)] \leq 1 - \delta.$$

In Boolean circuit complexity, one of the important question is about finding an explicit function $f : \{0,1\}^n \to \{0,1\}$ such that all circuits C of size at most $2^{-\epsilon n}$

$$\Pr_x[C(x) = f(x)] \leq \frac{1}{2} + 2^{-\delta n}.$$

for some constants $\epsilon, \delta > 0$.

We know that such functions exist by counting argument. But how do we find one explicitly? We will see in a couple of lectures that in order to get such an average case hard function, it is enough to find an explicit *worst-case* hard function. By the end of this lecture, we will reach very close to the goal.

Remark 1. *One may wonder why Impagliazzo's hard core lemma (see the previous lecture for the lemma statement) is not enough to get a $\frac{1}{2} - 2^{-\delta n}$-hard function starting with a $2^{-\delta n}$-hard function. The difficulty lies in the fact that the hard core lemma only tells about the* existence *of a hard core set such that on that set f is $\frac{1}{2} - 2^{-\delta n}$ -hard. We do not know how to sample from that hard core set efficiently.*

As a first step, we will show how to construct $1/n^2$ hard function starting from a worst-case hard function. More formally, we will prove the following theorem.

Theorem 3. *There exists an absolute constant $c \geq 1$ such that given $f : \{0,1\}^n \to \{0,1\}$ which is worst case hard for circuits of size s then one can construct a function $g : \{0,1\}^{O(n)} \to \{0,1\}$ which is $1/n^2$ hard for circuits of size $\frac{s}{n^c}$.*

Proof Strategy: The high level proof strategy is as follows. Given a truth table of function f we will construct a truth table of g. Suppose there exists a circuit of size s' which computes g correctly on at least $1 - 1/n^2$ fraction of the inputs then we will product another circuit $\tilde{C}$ of size at most $s' \cdot n^c$ which computes f exactly. This will finish the proof of the theorem.

1.1 Detour : Error correcting codes

For two strings $x, y \in \{0,1\}^m$ denote the hamming distance between x and y (which is just the fraction of locations where x and y differ) by $\Delta(x, y)$.

Definition 4 (ECC)**.** *Error correcting code is a subset of $\{0,1\}^m$ with the property that any two strings in the collection are far apart in the hamming distance. More formally, error correcting code can be thought of a mapping $E : \{0,1\}^n \to \{0,1\}^m$ with the property that for all $x, y \in \{0,1\}^n$ where $x \neq y$, $\Delta(x, y) \geq \delta m$. δ is called the distance of a code.*

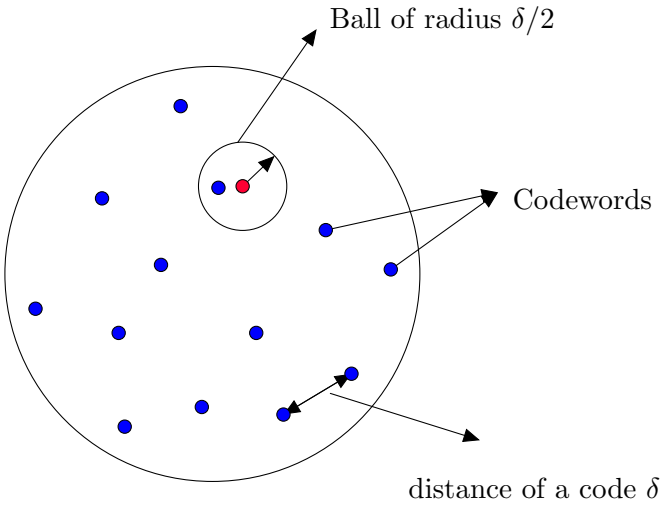

Figure 1: Error correcting code

Every two codewords are δ-far apart from each other. Therefore, given a word which is less than $\delta/2$ distance far apart form a codeword, there is a unique codeword which is in the ball of radius $\delta/2$ from that word. Decoding is procedure of finding a nearest codeword from a given string in $\{0,1\}^m$.

Definition 5 (Decoding)**.** *Given $w \in \{0,1\}^m$ and $\alpha > 0$, find $x \in \{0,1\}^n$ (if it exists) such that $\Delta(E(x), w) \leq \alpha m$.*

Let $\mathbb{F}$ be any finite field. Consider the following code. Let N be of the form d^m for some $d, m \in \mathbb{N}^+$. Encoding is a map which takes a string in $\mathbb{F}^N$ to $\mathbb{F}^M$ given as follows: Let $S \subseteq \mathbb{F}$ of size d. S^m is a subset of $\mathbb{F}^m$. Given a string $y \in \mathbb{F}^N$, view it as a function $u : S^m \to \mathbb{F}$ (using arbitrary but fixed one-to-one map from $[N]$ to S^m). Now consider a polynomial $P(x_1, x_2, \ldots, x_m)$ of degree at most $d-1$ in each variable that fits u (See Claim 6). Then the encoding of y is given as

$$E(y) = (P(z))_{z \in \mathbb{F}^m},$$

which is just the evaluation of polynomial P on $\mathbb{F}^m$.

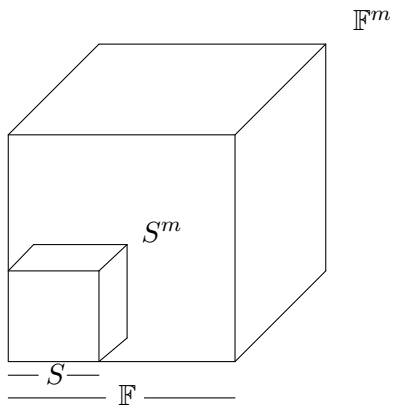

Figure 2: Encoding of a string

Claim 6. *Given any mapping $u : S^m \to \mathbb{F}$, there exists a polynomial P agreeing with u on S^m and degree of each variable in P is at most $d-1$.*

Proof. It is enough to show that for any $a \in S^m$ there exists a polynomial P_a with individual degree at most $d-1$ such that $P(a) \neq 0$ and $P(b) = 0$ for all $b \in S^m \setminus \{a\}$. Define P_a as

$$P_a(x_1, x_2, \ldots, x_m) = \prod_{i=1}^{m} \prod_{c \in S \setminus a_j} (x_i - c).$$

When $x = a$ all the terms are non-zero. Also, for any $b \in S^m \setminus \{a\}$ there is at least one term which evaluates to zero and hence $P_a(b) = 0$. Also, the way P_a is constructed, each x_i has its highest degree at most $d-1$.

To get the actual polynomial P which agrees with u, set

$$P(x_1, \ldots, x_m) = \sum_{a \in S^m} P_a(a)^{-1} u(a) P_a(x_1, \ldots, x_m).$$

It is easy to check that P satisfies the required conditions. □

1.1.1 Local decoding

The important property of the above code is that given a string which is not far from a particular encoding of a string x in $\mathbb{F}^N$, we can recover the value of x at any location $i \in [N]$ by probing few locations in the received word. This property will crucially get used in proving Theorem 3.

Goal: Given access to $w : \mathbb{F}^m \to \mathbb{F}$ such that there exists a polynomial P of degree $d-1$ in each variable with the property that

$$\Pr_{z \in \mathbb{F}^m}[w(z) = P(z)] \geq 1 - \alpha$$

Then, given a point $a \in S^m$, find $P(a)$.

Consider the following algorithm:

Local Decoding Algorithm:

Given : $a \in \mathbb{F}^m$ and access to $w : \mathbb{F}^m \to \mathbb{F}$.
Output : $P(a)$, where P is such that $\Pr_{z \in \mathbb{F}^m}[w(z) = P(z)] \geq 1 - \alpha$ and degree of each variable in P is at most $d-1$.

1. Pick $b \in \mathbb{F}^m$ u.a.r.
2. Look at the line $\{a + bt | t \in \mathbb{F}\} \subseteq \mathbb{F}^m$.
3. Query $w(a + bt)$ for each $t \in \mathbb{F} \setminus \{0\}$.
4. If there is a degree at most $m(d-1)$ univariate polynomial $\tilde{Q}(t)$ such that $\tilde{Q}(t) = w(a + bt)$ for all $t \in \mathbb{F} \setminus \{0, \}$, then output $\tilde{Q}(0)$. Otherwise, output 0.

Analysis of the Algorithm: We now analyze the probability that the algorithm outputs a correct value for $P(a)$.

Lemma 7. *Let $|\mathbb{F}| \geq md$, the probability that the algorithm outputs the correct value is at least $1 - |\mathbb{F}|\alpha$.*

Proof. For any fixed $t \in \mathbb{F} \setminus \{0\}$, $a + tb$ is uniformly distributed in $\mathbb{F}^m$ as $b \in \mathbb{F}^m$ u.a.r. Thus, the guarantee in w gives

$$\Pr_b[w(a + tb) \neq P(a + tb)] \leq \alpha.$$

By union bound,

$$\Pr_b[\exists t \in \mathbb{F} \setminus \{0\} \text{ s.t. } w(a + tb) \neq P(a + tb)] \leq (|\mathbb{F}| - 1)\alpha \leq |\mathbb{F}|\alpha$$

If for every $t \in \mathbb{F} \setminus \{0\}$, $w(a + tb) = P(a + tb)$ then the algorithm will find $\tilde{Q}$ of low degree and in that case $\tilde{Q}(t) = P(a + tb)$. This is because $P(a + tb)$ and $\tilde{Q}(t)$ both are univariate polynomials of degree at most $m(d-1)$ and agree on $|\mathbb{F}| - 1 > m(d-1)$ many points. Hence, they must be the same polynomial. Thus,

$$\Pr_b[\text{ The algorithm outputs } P(a)] \geq 1 - |\mathbb{F}|\alpha.$$

□

Lemma 8. *If $\alpha < \frac{1}{3|\mathbb{F}|}$ then there exists a* deterministic algorithm *such that given any $a \in S^m$ it outputs $P(a)$.*

Proof. For $\alpha < \frac{1}{3|\mathbb{F}|}$, from Lemma 7 the success probability is at least $2/3$. By repeating the algorithm multiple times, say $\texttt{poly}(|\mathbb{F}|, m)$, and taking the majority we can make the success probability at least $1 - |\mathbb{F}|^{-m}$. As the total number of possible query points are $|S^m| < |\mathbb{F}|^m$, there exists a fixing of a random seed to the algorithm such that the algorithm works for all inputs and outputs correct value of $P(a)$ for all $a \in S^m$. □

Remark 2. *Given access to w in the form of a circuit C (over field $\mathbb{F}$), one can construct a circuit $\tilde{C}$ whose size is $\texttt{size}(C) \cdot \texttt{poly}(|\mathbb{F}|, m)$ such that $\tilde{C}$ computes P on S^m exactly. This follows essentially from the fact that the interpolation step (Step 4) can be carried out with a circuit of size $\texttt{poly}(|\mathbb{F}|, m)$.*

1.2 Back to the proof of Theorem 3

Proof. We switch from Boolean circuits with AND/OR gates to circuits over finite field $\mathbb{F}$ where the gates are additions and multiplications over $\mathbb{F}$. We can convert circuits over large field to Boolean circuits by associating some kind of mapping from field elements to Boolean strings. The conversion can be made with a blow-up of at most $\texttt{poly}(|\mathbb{F}|)$ factor which will be insignificant for the proof of the theorem.

Let $\mathbb{F}$ be a large field (we'll set its size in the end). Let's map arbitrarily $\{0,1\}$ to two of the field elements $\mathbb{F}$ and let's call them $\{0,1\}$ for convenience. Let d and m be positive integers such that $d^m = 2^n$. Let ϕ be any efficiently computable one-to-one mapping $\phi : S^m \to \{0,1\}^n$. Given a Boolean function $f : \{0,1\}^n \to \{0,1\}$, consider the encoding of a function $P_f : S^m \to \mathbb{F}$, where $P_f(x) = f(\phi(x))$. Denote that encoded function by $g : \mathbb{F}^m \to \mathbb{F}$. We will show that g is average case hard for circuits of large size.

Suppose C is a circuit of size s' which computes g on at least $1 - \alpha$ fraction of the inputs where $\alpha = 1/n^2$.

$$\Pr_z[C(z) = g(z)] \geq 1 - \alpha.$$

This is the same set-up as the local decoding problem. Given access to C, we can construct another circuit $\tilde{C}$ which computes P_f exactly on S^m and the size of $\tilde{C}$ is at most $\texttt{size}(\tilde{C}) \leq \texttt{size}(C) \cdot \texttt{poly}(m, |\mathbb{F}|)$. Thus, there exists a circuit of size at most $\texttt{size}(C) \cdot \texttt{poly}(m, |\mathbb{F}|)$ which computes f exactly which gives a contradiction unless $\texttt{size}(C) \cdot \texttt{poly}(m, |\mathbb{F}|) < s$ We now set the parameters: We need $\alpha < 1/3|\mathbb{F}|$ for the decoding algorithm to work and $\mathbb{F}^m = 2^{O(n)}$ to maintain the number of input bits to g linear in n. We also need $m(d-1) < |\mathbb{F}|$ for Step 4 to work. Setting $d = n$, $m = \frac{n}{\log n}$, $|\mathbb{F}| = 3md$ and $\alpha = \frac{1}{n^2}$ satisfy all the conditions. Thus, we get that size of C is bounded by

$$\texttt{size}(C) \cdot \texttt{poly}(m, |\mathbb{F}|) > s$$

for it to compute f exactly. Hence, $\texttt{size}(C)$ has to be at least $s/\texttt{poly}(m, |\mathbb{F}|) = s/n^c$, for some fixed constant c, for it to compute g on at least $1 - 1/n^2$ fraction of inputs. □

Observation 9. *The construction shows that if $f \in \texttt{PSPACE}$ then $g \in \texttt{PSPACE}$. Thus, if $\texttt{PSPACE}$ is not in $P \setminus \texttt{poly}$ then there is a $1/n^2$ -hard function in $\texttt{PSPACE}$.*

2 Yao's XOR Lemma

In this section, we will show how to construct a function that is $\approx 1/2$ hard for circuits of sub-exponential size. The main trick in getting this kind of amplification is so called XOR*ing* trick from Yao.

Lemma 10 (Yao's XOR lemma)**.** *If f be any δ-hard function for circuits of size s. Define a function $g : (\{0,1\}^n)^k \to \{0,1\}$ as follows*

$$g(x_1, x_2, \ldots, x_k) = \bigoplus_{i=1}^{k} f(x_i).$$

Then, g is $\frac{1}{2} - \frac{1}{2}(1-2\delta)^k - \epsilon$ hard for circuits of size $\mathtt{poly}(\epsilon, \delta, \frac{1}{n}) \cdot s$.

Before proving the lemma, let's see that the lemma is essentially tight.

Tightness: Consider a biased coin such that $\Pr[\text{heads}] = \delta$ with $\delta < 1/2$. For a single coin toss, there is an algorithm which predicts the coin toss with probability $1 - \delta$ - always output tails. But consider predicting whether in k coin tosses if there are even number of heads.

$$\begin{aligned}
\Pr[\text{Even number of heads in } k \text{ tosses}] &= \sum_{i=0}^{\lfloor k/2 \rfloor} \binom{n}{2i} \delta^{2i} (1-\delta)^{k-2i} \\
&= \frac{1}{2}\left(((1-\delta)+\delta)^k + ((1-\delta)-\delta)^k\right) \\
&= \frac{1}{2} + (1-2\delta)^k
\end{aligned}$$

Thus, no algorithm can predict if there are even number of heads in k coin tosses better than $\frac{1}{2} + (1-2\delta)^k$ probability.

Remark 3. *It is not known whether the function g in Lemma 10 is hard for circuits of size $\approx k \cdot s$*

Proof of Lemma 10. Suppose C is a circuit of size s' such that

$$\Pr_{x_1, x_2, \ldots, x_k}\left[C(x_1, x_2, \ldots, x_k) = \bigoplus_{j=1}^{k} f(x_j)\right] \geq \frac{1}{2} + (1-2\delta)^k + \epsilon. \tag{1}$$

Let H be a hard core of f. We know from the hard core lemma that $|H| \geq 2\delta \cdot 2^n$. Thus, for all circuits $\tilde{C}$ of size $\mathtt{poly}(\epsilon, \delta, \frac{1}{n}) \cdot s$

$$\Pr_{x \in H}\left[f(x) = \tilde{C}(x)\right] \leq \frac{1}{2} + \frac{\epsilon}{2}. \tag{2}$$

The size of H implies that

$$\Pr_{x_1, x_2, \ldots, x_k}[\text{ all } x_i \notin H] \leq (1-2\delta)^k. \tag{3}$$

From (1) and (3), we can conclude that

$$\Pr_{x_1,x_2,\ldots,x_k}\left[C(x_1,x_2,\ldots,x_k)=\bigoplus_{j=1}^{k} f(x_j) \wedge \text{ some } x_j \in H\right] \geq \frac{1}{2}+\epsilon$$

Thus there exists $i \in [k]$ such that

$$\Pr_{x_1,x_2,\ldots,x_k}\left[C(x_1,x_2,\ldots,x_k)=\bigoplus_{j=1}^{k} f(x_j) \mid x_i \in H\right] \geq \frac{1}{2}+\epsilon$$

From above, there exists setting of $x_1, x_2, \ldots, x_{i-1}, x_{i+1}, x_{i+2}, \ldots, x_k$ such that

$$\Pr_{x_i \in H}\left[C(x_1,x_2,\ldots,x_{i-1},\mathbf{x_i},x_{i+1},x_{i+2},\ldots,x_k)=\bigoplus_{j=1}^{i-1} f(x_j) \oplus \mathbf{f(x_i)} \oplus \bigoplus_{j=i+1}^{k} f(x_j)\right] \geq \frac{1}{2}+\epsilon$$

Letting $\bigoplus_{j=1}^{i-1} f(x_j) \bigoplus_{j=i+1}^{k} f(x_j) = b$ for some $b \in \{0,1\}$ we get

$$\Pr_{x_i \in H}[C(x_1,x_2,\ldots,x_{i-1},\mathbf{x_i},x_{i+1},x_{i+2},\ldots,x_k)=\mathbf{f(x_i)} \oplus b] \geq \frac{1}{2}+\epsilon$$

which contradicts (2) unless $s' > \texttt{poly}(\epsilon, \delta, \frac{1}{n}) \cdot s$. □

3 Conclusion

Let us combine both the hardness amplification theorems to see what we get from a worst case hard function.

Start with f_1 which worst case hard for circuits of size $s_1 = 2^{n^{3/4}}$. By Theorem 3, we can construct a function f_2 on $n' = O(n)$ bits which is $1/n^2$ hard for circuits of size $s_2 = 2^{O(n^{3/4})}$ Set $k = O(n^{10})$ and $\epsilon = 2^{-\sqrt{n}}$. If g is a k-XOR of f_2 then by Yao's XOR Lemma, g is $\frac{1}{2} - \left(1 - \frac{1}{n^2}\right)^{O(n^{10})} - 2^{-\sqrt{n}}$ - hard for circuits of size $2^{O(\sqrt{n})} \cdot s_2 = 2^{O(n^{3/4})}$. Letting $N = n'^{11}$, g is a function on N bits which is $\frac{1}{2} - 2^{-N^{0.01}}$- hard for circuits of size $2^{N^{0.1}}$.

In the next lecture, we will see starting with a worst case hard function for circuits of exponential size, how to construct a function on n bits which is $\frac{1}{2} - 2^{-\epsilon n}$ hard for circuits of size $2^{\delta n}$ for some constants $\epsilon, \delta > 0$.